APPENDIX A:

```perl
#! /usr/bin/perl -w
# camspool-backend.pl
#==============================================================================
v
#
####BSTDHDR####
#
#==============================================================================
=
#
####DESCRIPTIONBEGIN####
#  AUTHOR(S):      Matthew H. Gerlach
#  PURPOSE:
#  DESCRIPTION:
#
#      This is a perl script that implements the "backend" portion
#  of the camspooler.  The backend is responsible for taking the
#  the pictures that have been uploaded to the camspool frontend
#  and in turn uploading it to the real Lightsurf web server to
#  a particular user's account.
#
#      This particular implementation simply involves polling a directory,
#  looking for new files that have been downloaded.  When it finds one,
#  it reads the pictures ancillary data file for information about the
#  picture and its associated user, or if a new account needs to be made.
#  With the information, it tries to deposit the pictures in the correct
#  account.  Basically, once a picture has be put in the directory by the
#  camspool front end, the backend will try until hell freezes over to
#  get that picture to the account.
#
#
#
####DESCRIPTIONEND####
#==============================================================================
=
#
####COPYRIGHTBEGIN####
#
#  (c) Copyright 1999, 2000 Lightsurf Technologies, Inc.  ALL RIGHTS RESERVED.
#
#
####COPYRIGHTEND####
#==============================================================================
=
#
####ESTDHDR####
#==============================================================================
^

use strict;
use LS_UnixDaemonUtils;
use LS_UploadClient;
use XML::Simple;
use Data::Dumper;
use LWP::UserAgent;
use Benchmark;
```

1

```perl
if ($#ARGV < 1)
{
    &print_usage;
}

my($picture_dir) = shift;

if (! -d $picture_dir)
{
    LS_LogPrint "$picture_dir is not a directory\n";
    &print_usage;
}

my(%url_info) = ();

$url_info{start} = shift;

if (!defined($url_info{start}))
{
    LS_LogPrint "you must give a url to upload to\n";
    &print_usage;
}

my($sleep_time) = 10;
my($arg);
my($log_filename) = undef;
my($pid_filename) = undef;
my $uid = undef;
my $gid = undef;

while ($arg = shift)
{
    if ($arg =~ /^--sleep$/)
    {
        $arg = shift;
        if (!defined($arg))
        {
            LS_LogPrint "-t needs a time value\n";
            &print_usage;
        }
        $sleep_time = oct($sleep_time) if $sleep_time =~ /^0/;
    }
    elsif ($arg =~ /^-D$/)
    {

        $pid_filename = shift;
        if (!defined($pid_filename))
        {
            LS_LogPrint "-D needs a pid_filename\n";
            &print_usage;
        }
    }
    elsif ($arg =~ /^--log$/)
    {
        $log_filename = shift;
        if (!defined($log_filename))
```

2

```perl
            {
                LS_LogPrint "--log needs a log_filename\n";
                &print_usage;
            }
        }
        elsif ($arg =~ /^--uid$/)
        {
            $uid = shift;
            if (!defined($uid))
            {
                LS_LogPrint "--uid needs a uid\n";
                &print_usage;
            }
        }
        elsif ($arg =~ /^--gid$/)
        {
            $gid = shift;
            if (!defined($gid))
            {
                LS_LogPrint "--uid needs a uid\n";
                &print_usage;
            }
        }
        elsif ($arg =~ /^--login$/)
        {
            $url_info{login} = shift;
            if (!defined($url_info{login}))
            {
                LS_LogPrint "--login needs a login\n";
                &print_usage;
            }
        }
        elsif ($arg =~ /^--passwd$/)
        {
            $url_info{passwd} = shift;
            if (!defined($url_info{passwd}))
            {
                LS_LogPrint "--passwd needs a passwd\n";
                &print_usage;
            }
        }
        elsif ($arg =~ /^--del$/)
        {
            $url_info{del_photo} = 1;
        }
        else
        {
            LS_LogPrint "Unknown option: $arg\n";
            &print_usage;
        }
    }

    if (defined($log_filename))
    {
        LS_SetLogFile($log_filename);
    }

    if (defined($pid_filename))
```

```perl
{
    &LS_BecomeDaemon();

    $SIG{TERM} = \&HandleSigTerm;

    &LS_WritePidFile($pid_filename);

}

if (defined($gid))
{
    LS_SetGid($gid);
}

if (defined($uid))
{
    LS_SetUid($uid);
}

#
# write pid file after changing uid/gid
# so that we can delete pid file if necessary
#
if (defined($pid_filename))
{
    $SIG{TERM} = \&HandleSigTerm;

    &LS_WritePidFile($pid_filename);

}

select(STDERR); $| = 1;
select(STDOUT); $| = 1;

&CamspoolBackend($picture_dir, $sleep_time, \%url_info);

sub print_usage
{
    print "Usage $0 <picture_dir> <url> [--sleep sleep_interval ]\n";
    print "                              [-D pid_file ]\n";
    print "                              [--log log_file ]\n";
    print "                              [--uid uid ]\n";
    print "                              [--login login ]\n";
    print "                              [--passwd passwd ]\n";
    print "                              [--del ]\n";
    print "     picture_dir - directory to poll for tagged pictures\n";
    print "     url         - url of server to upload to (e.g.
http://dsheth-nt4:8080)\n";
    print "     --sleep     - seconds between directory polls (default =
10)\n";
    print "     -D          - start process as daemon and write the rsulting
prociess it to pid_file \n";
    print "     --log       - LSLogPrint log info to passed file (default =
STDOUT)\n";
    print "     --uid       - set program's effective user id\n";
    print "     --login     - optional login used for server authentication\n";
    print "     --passwd    - optional login used for server authentication\n";
    print "     --del       - delete photos when uploaded to server\n";
```

4

```perl
        exit(1);
    }


    sub HandleSigTerm
    {
        LS_LogPrint "Terminated\n";
        unlink($pid_filename);
        exit(1);
    }


    #
    # CamspoolBackend
    #
    #    This function implements the basic flow control
    # of the Campspool back end.  In short it runs forever,
    # periodically checking the passed directory for any
    # upload jobs (i.e. files ending with ".tag").  As long as it
    # finds jobs to upload, it will; otherwise, it sleeps for a bit
    # before checking for more jobs.
    #
    sub CamspoolBackend
    {
        my($picture_dir, $sleep_time, $url_ref) = @_;
        my(@jobs);
        my($job_file_name, $job, $upload_client);

        my $last_ticket = "";

        my $jobs_completed = 0;

        my $job_successful;

        LS_LogPrint "Starting $0\n";

        while (1)
        {
            @jobs = &getUploadJobs($picture_dir);


            $jobs_completed = 0;
            foreach $job_file_name (@jobs)
            {
                $job = ReadJob("$picture_dir/$job_file_name.tag");

                if (!defined($job))
                {
                    LS_LogPrint "Failed to parse job
$picture_dir/$job_file_name\n";
                    &renameBadJob("$picture_dir/$job_file_name");
                    next;
                }

                LS_LogPrint "\n";
                LS_LogPrint "Attempting upload job $job_file_name\n";
                $job_successful = 0;
```

5

```perl
    #
    # given a device_login (a.k.a. a ticket) we need to create
    # an upload client.  As long we are uploading jobs with
    # the ticket, we can reuse the upload client.
    #
    if ($job->{ticket} ne $last_ticket)
    {
        if (defined($url_ref->{login}))
        {
            $upload_client = new LS_UploadClient($url_ref->{start},
$job->{ticket},
                                                        login  =>
$url_ref->{login},
                                                        passwd =>
$url_ref->{passwd},);
        }
        else
        {
            $upload_client = new LS_UploadClient($url_ref->{start},
$job->{ticket});
        }


        if (!defined($upload_client) || !ref($upload_client))
        {
            LS_LogPrint "Error creating upload client for job
$job_file_name\n";
            &renameBadJob("$picture_dir/$job_file_name");
            $last_ticket = "";
            next;
        }

        # if we fall through here, then we successfully got an
upload_client
        }

        if ($job->{type} eq "image/x-lspp")
        {
            $job_successful =
$upload_client->UploadImageCompartment($job->{guid},

$job->{type},

$job->{savedFilename},

$job->{part}, 0, 0);
        }
        elsif ($job->{type} eq "image/x-lsanc")
        {
            $job_successful = &CamspoolUploadAnc($picture_dir, $job,
$upload_client);

            if (!defined($job_successful))
            {
                LS_LogPrint "Anc job $job_file_name is empty\n";
                $job_successful = 1;
            }
```

6

```perl
            }
            else
            {
                LS_LogPrint "Unknown job type, $job->{type}, in
$job_file_name\n";
            }

            if ($job_successful > 0)
            {
                LS_LogPrint sprintf "Successfully uploaded %s job
$job_file_name\n", $job->{type};
                $jobs_completed++;
                $last_ticket = $job->{ticket};

                if (exists($url_ref->{del_photo}))
                {
                    if (!unlink ("$job->{savedFilename}"))
                    {
                        LS_LogPrint "Failed to delete job data
$job->{savedFilename}: $!\n";
                        &renameBadJob("$picture_dir/$job_file_name");
                    }
                    elsif (!unlink ("$picture_dir/$job_file_name.tag"))
                    {
                        LS_LogPrint "Failed to delete job
$picture_dir/$job_file_name: $!\n";
                        &renameBadJob("$picture_dir/$job_file_name");
                    }
                }
                else
                {
                    if (!rename("$picture_dir/$job_file_name.tag",
"$picture_dir/$job_file_name.snt"))
                    {
                        LS_LogPrint sprintf "Failed to rename %s to %s: $!\n",
                                            "$picture_dir/$job_file_name.tag",
                                            "$picture_dir/$job_file_name.snt"
;

                    }
                }
            }
            else
            {
                LS_LogPrint sprintf "Failed to upload %s job
$job_file_name\n", $job->{type};
                &renameBadJob("$picture_dir/$job_file_name");
                $last_ticket = "";
            }

        }

        #
        # if there are no new jobs or we couldn't successfully upload any job
        # then we will sleep a bit to give the server some breathing room
        #
        if ($jobs_completed <= 0)
        {
```

7

```perl
            sleep $sleep_time;
        }
    }
}


#
#   getUploadJobs
#
#     This function will return a list of upload jobs.  These
#     jobs are just files in the passed dir, ending with ".tag".
#
sub getUploadJobs
{
    my($dir) = @_;
    my(@tagged_files);

    if (!opendir(JOBS , "$dir"))
    {
        LS_LogPrint "can't open directory $dir: $!\n";
        exit(1);
    }

    @tagged_files = readdir JOBS;

    if (!closedir JOBS)
    {
        LS_LogPrint "can't close directory $dir: $!\n";
        exit(1);
    }

    @tagged_files = grep {s/\.tag$//} @tagged_files;

    return(sort @tagged_files);
}

sub ReadJob
{
    my ($filename) = @_;

    if (!open(FILE, $filename))
    {
        LS_LogPrint "Failed to open job file, $filename: $!\n";
        return undef;
    }

    my $line;
    my ($key, $value, %hash);

    while ($line = <FILE>)
    {
        chomp $line;

        ($key, $value) = split(/ /, $line);
        $hash{$key} = $value;
    }

    if (!close(FILE))
```

8

```perl
    {
        LS_LogPrint "Failed to close job file, $filename: $!\n";
        return undef;
    }

    return \%hash;
}
#
# renameBadJob
#
# For whatever reason we might encounter a job file that broken in some
# way.  To avoid continually retrying the bad job, we rename the filename
# so that it ends with ".bad".
#
sub renameBadJob
{
    my ($job_filename) = @_;

    if (!rename("$job_filename.tag", "$job_filename.bad"))
    {
        LS_LogPrint "Failed to rename bad job,$job_filename: $!\n";
    }
}

sub CamspoolUploadAnc
{
    my ($picture_dir, $job, $upload_client) = @_;

    my($old_RS) = $/;
    my($input);

    my($TITLE, $LOCATION, $COMMENTS);
    my(@EMAIL );
    my($recip, $addr);

    my $anc_file = "$job->{savedFilename}";

    $TITLE     = "";
    $LOCATION  = "";
    $COMMENTS  = "";
    @EMAIL     = ();


    $/ = undef;  # set input separator to undef to read whole file

    if (!open(ANC_FILE, $anc_file))
    {
        LS_LogPrint "failed to open anscillary data file $anc_file\n";
        $/ = $old_RS;
        return(0);
    }
    $input = <ANC_FILE>;
    $input =~ s|<Ver>['\n\r]*.*['\n\r]*</Ver>||i;

    my $xml = XMLin($input, forcearray => 1);

    close(ANC_FILE);
    $/ = $old_RS;
```

9

```perl
        ## Get the photo title
        defined $xml->{title}->[0] && ($TITLE=$xml->{title}->[0]);

        ## ...Location...
        defined $xml->{location}->[0] && ($LOCATION=$xml->{location}->[0]);

        ## ...Comments...
        defined $xml->{comments}->[0] && ($COMMENTS=$xml->{comments}->[0]);

        ## ...email list...
        if (exists $xml->{email_list}->[0]->{email})
        {
            my $email=$xml->{email_list}->[0]->{email};
            foreach $recip (@$email)
            {
              my $alias="",$addr="";
              defined $recip->{alias}->[0] && ($alias=$recip->{alias}->[0]);
              defined $recip->{emailadd}->[0] && ($addr=$recip->{emailadd}->[0]);
              push @EMAIL,("$addr");
            }
        }

        my $rval = undef;

        if ((length($TITLE)    > 0) || (length($LOCATION) > 0) ||
            (length($COMMENTS) > 0)    )
        {
#         LS_LogPrint "title is \"$TITLE\"\ncomments are
\"$COMMENTS\"\nlocataion is \"$LOCATION\"\n";

            if (!$upload_client->SetMetaData( "guid", $job->{guid},
                                              title    => $TITLE,
                                              location => $LOCATION,
                                              comments => $COMMENTS, ))
            {
                return(0);
            }
            else
            {
                $rval = 1;
            }
        }

        if ($#EMAIL >= 0)
        {
            return($upload_client->ShipEmailAddrs("guid", $job->{guid}, \@EMAIL));
        }
        return($rval);
}

#! /usr/bin/perl -w
# camspool-frontend.pl
#=======================================================================
v
#
####BSTDHDR####
#
```

```
#==============================================================================
=
#
####DESCRIPTIONBEGIN####
#  AUTHOR(S):      Matthew H. Gerlach
#  PURPOSE:        Implements Camspool's front end
#  DESCRIPTION:
#
#  This program implementes the Lightsurf Camspool's frontend.
#  It is responsible for receiving (or initiating) TCP connections from/to
#  a Lightsurf camera.  Once the TCP connection is established,
#  this program becomes a "client" in terms of making a series
#  of UICHAN resqests to the camera to get information and utlimately
#  fetching pictures to a local harddrive.
#
#
####DESCRIPTIONEND####
#==============================================================================
=
#
####COPYRIGHTBEGIN####
#
#  (c) Copyright 1999, 2000 Lightsurf Technologies, Inc.  ALL RIGHTS RESERVED.
#
#
####COPYRIGHTEND####
#==============================================================================
=
#
#
####ESTDHDR####
#==============================================================================
^
use strict;
use IO::Socket;
use Data::Dumper;
use LS_UnixDaemonUtils;
use LS_Uichan;
use LS_UploadClient;
use POSIX "sys_wait_h";

my $gDefaultTcpPort = 13002;

if ($#ARGV < 0)
{
    &print_usage();
}

my %gSpoolInfo = ();

$gSpoolInfo{local_dir} = shift;
$gSpoolInfo{guid_type} = "tagged";

if (! -d $gSpoolInfo{local_dir})
{
    die "    $gSpoolInfo{local_dir} is not a directory\n";
}
```

11

```perl
my $arg;
my $tcp_port      = $gDefaultTcpPort;
my $log_filename = undef;
my $host          = undef;
my $pid_filename = undef;
my $uid           = undef;
my $gid           = undef;
while ($arg = shift)
{
    if ($arg =~ /^-D$/)
    {

        $pid_filename = shift;
        if (!defined($pid_filename))
        {
            print "-D needs a pid_filename\n";
            &print_usage;
        }
    }
    elsif ($arg =~ /^--log$/)
    {
        $log_filename = shift;
        if (!defined($log_filename))
        {
            print "--log needs a filename\n";
            &print_usage;
        }
    }
    elsif ($arg =~ /^--uid$/)
    {
        $uid = shift;

        if (!defined($uid))
        {
            print "--uid needs a uid\n";
            &print_usage;
        }

    }
    elsif ($arg =~ /^--gid$/)
    {
        $gid = shift;

        if (!defined($gid))
        {
            print "--gid needs a gid\n";
            &print_usage;
        }

    }   elsif ($arg =~ /^--login$/)
    {
        $gSpoolInfo{login} = shift;
        if (!defined($gSpoolInfo{login}))
        {
            print "--login needs a login\n";
            &print_usage;
        }
```

```perl
        }
        elsif ($arg =~ /^--passwd$/)
        {
            $gSpoolInfo{passwd} = shift;
            if (!defined($gSpoolInfo{passwd}))
            {
                print "--passwd needs a passwd\n";
                &print_usage;
            }

        }
        elsif ($arg =~ /^--host$/)
        {
            $host = shift;
            if (!defined($host))
            {
                print "--host needs a host_id\n";
                &print_usage;
            }
        }
        elsif ($arg =~ /^--port$/)
        {
            $tcp_port = shift;
            if (!defined($tcp_port))
            {
                print "--port needs a tcp_port\n";
                &print_usage;
            }
        }
        elsif ($arg =~ /^--del$/)
        {
            $gSpoolInfo{del_photo} = 1;
        }
        elsif ($arg =~ /^--all$/)
        {
            $gSpoolInfo{guid_type} = "all";
        }
        elsif ($arg =~ /^--url$/)
        {
            $gSpoolInfo{url} = shift;
            if (!defined($gSpoolInfo{url}))
            {
                print "--url needs a url\n";
                &print_usage;
            }
        }
        else
        {
            print "Unknown option: $arg\n";
            &print_usage;
        }

    }

if (defined($log_filename))
{
    LS_SetLogFile($log_filename);
}
```

13

```perl
autoflush STDERR 1;
autoflush STDOUT 1;

if (defined($host))
{
    &CamspoolConnector($host, $tcp_port, \%gSpoolInfo);
}
else
{
    &CamspoolListener($tcp_port, \%gSpoolInfo, $pid_filename, $uid, $gid);
}

exit(0);



sub print_usage
{
    print "Usage $0 <local_dir> \n";
    print "                                        [--url sync_url ]\n";
    print "                                        [-D pid_file ]\n";
    print "                                        [--log log_file ]\n";
    print "                                        [--uid uid ]\n";
    print "                                        [--gid gid ]\n";
    print "                                        [--login login ]\n";
    print "                                        [--passwd passwd ]\n";
    print "                                        [--host host_id ]\n";
    print "                                        [--port tcp_port ]\n";
    print "                                        [--del ]\n";
    print "                                        [--all ]\n";
    print "     local_dir  - local directory to store pictures\n";
    print "     url        - url of server to perform database sync query\n";
    print "     -D         - start process as daemon writing to log_file and
pid_file \n";
    print "     --log      - send output to log_file instead of STDOUT\n";
    print "     --uid      - set program's effective user id\n";
    print "     --gid      - set program's effective group id\n";
    print "     --login    - optional login used for server authentication\n";
    print "     --passwd   - optional passwd used for server authentication\n";
    print "     --host     - initiate TCP connection to host_id, otherwise
listen for TCP connections\n";
    print "     --port     - either listen or connect to tcp_port, default =
$gDefaultTcpPort\n";
    print "     --del      - delete photos on camera instead of setting state
to \"done\"\n";
    print "     --all      - fetch all pictures rather than just \"tagged\"
pictures\n";
    exit(1);
}

#
#  This little function handles a SIG_TERM signal.
#  it just removes the $pid_filename and exits
#
sub HandleSigTerm
{
    LS_LogPrint "Terminated\n";
    unlink($pid_filename);
```

14

```perl
    exit(1);
}

#
#   This function implements a Camspool connector.
#   In other words it initiates a TCP connection
#   to a camera and then has a standard CamspoolSession.
#
sub CamspoolConnector
{
    my($host, $port, $info_ref) = @_;

    my $sock = new IO::Socket::INET(PeerAddr => $host,
                                    PeerPort => $port,
                                    Proto    => 'tcp', );
    if (!defined($sock))
    {
        LS_LogPrint "Failed to connect to $host:$port\n    $!\n";
        exit(1);
    }

    my $uichan = new LS_Uichan($sock);

    $info_ref->{session_id} = 0;
    $info_ref->{port}       = $port;

    &CamspoolSession($uichan, $info_ref);

    $uichan->Empty();

    close $sock;
}

#
#   REAPER
#
#   Since CamspoolListener() forks children for each incoming connection,
#   the children must be reaped when they die.  This little
#   function was taken right from Chapter 6 of "Programming Perl" 2nd Edition.
#
sub REAPER
{
    $SIG{CHLD} = \&REAPER;

    while (waitpid(-1, WNOHANG) > 0)
    {
    }
}

#
# This function implements a Camspool listener.
# Forever, this function will accept TCP connections,
# forks, and has the child perform a stand CamspoolSession.
#
sub CamspoolListener
{
    my($port, $info_ref, $pid_filename, $uid, $gid) = @_;
```

15

```
LS_LogPrint "Starting $0\n";

my $sock = new IO::Socket::INET(LocalPort => $port,
                                Proto     => 'tcp',
                                Reuse     => 1,
                                Listen    => SOMAXCONN,);

if (!defined($sock))
{
    LS_LogPrint "Failed to create listening socket: $!\n";
    exit(1);
}

#
# don't bother becoming a daemon until we know
# we can bind to the socket.
#
if (defined($pid_filename))
{
    LS_BecomeDaemon();

    $SIG{TERM} = \&HandleSigTerm;

    LS_WritePidFile($pid_filename);
}

#
# we must hold off setting the gid/uid until
# we have bound to the socket.  This allows
# root to bind to a priveledge port, and then
# become a nobody.  Be sure to set gid before
# uid.
#
if (defined($gid))
{
    LS_SetGid($gid);
}

if (defined($uid))
{
    LS_SetUid($uid);
}

#
# we must write the pid file after we switch uid
# so that we can delete when we get terminated
#
if (defined($pid_filename))
{
    $SIG{TERM} = \&HandleSigTerm;

    LS_WritePidFile($pid_filename);
}

my ($new_sock, $child_pid);

my $session_counter = 0;
```

16

```perl
    $SIG{CHLD} = \&REAPER;

    while (1)
    {
        LS_LogPrint "Waiting for connection\n";

        $new_sock = $sock->accept();

        if (!defined($new_sock))
        {
            LS_LogPrint "Accept failed: $!\n";
            next;
        }

        $session_counter++;

        $child_pid = fork();

        if (!defined($child_pid))
        {
            LS_LogPrint "fork failed: $!\n";
            close($new_sock);
            next;
        }

        if ($child_pid == 0)
        {
            # Child closes its copy of the main socket
            close $sock;

            LS_LogPrint sprintf "Accepted connection from %s:%d\n",
                                $new_sock->peerhost(),
                                $new_sock->peerport();

            my $uichan = new LS_Uichan($new_sock);

            $info_ref->{session_id} = $session_counter;
            $info_ref->{port}       = $port;

            &CamspoolSession($uichan, $info_ref);

            $uichan->Empty();

            close $new_sock;

            exit(0);
        }
        else
        {
            # Parent closes copy of child's socket.
            close $new_sock;
        }
    }
}

#
# CamspoolSession
#
```

17

```perl
# This function performs a single "syncing" session with
# a camera.  A session lasts as long as there are
# pictures to be fetched and nothing fails.
#
sub CamspoolSession
{
    my($uichan, $info_ref) = @_;

    if (!$uichan->AuthCamera())
    {
        LS_LogPrint "Could Not Authenticate Camera\n";
        exit(1);
    }

    LS_LogPrint "Successfully Authenticated Camera\n";

    my $ticket = $uichan->GetTicket();

    if (!defined($ticket))
    {
        LS_LogPrint "Could not get Ticket\n";
        exit(1);
    }

    LS_LogPrint "Got ticket $ticket\n";

    my @tagged_guids;
    my @sync_guids;
    my $upload_client = undef;

    $info_ref->{session_count} = 0; # initialize count of files moved during
session
    $info_ref->{byte_count}    = 0;
    my $picture_count          = 0;

    my $session_start_time = time;
    my $list_ref_to_fetch;
    my $done = 0;
    while (!$done)
    {
        @tagged_guids = ();
        @sync_guids   = ();

        if (!$uichan->GetGuids($info_ref->{guid_type}, \@tagged_guids))
        {
            LS_LogPrint "Could not get tagged guids\n";
            last;
        }

        if ($#tagged_guids < 0)
        {
            LS_LogPrint "Session cleanly ended\n";
            last;
        }

        if (!exists($info_ref->{url}))
        {
            $list_ref_to_fetch = \@tagged_guids;
```

18

```perl
        }
        else
        {
            #
            #   Make db transaction to determine subset of @tagged_guids that
should
            #   be uploaded.
            #
            if (!defined($upload_client))
            {
                if (exists($info_ref->{login}) && exists($info_ref->{passwd}))
                {
                    $upload_client = new LS_UploadClient($info_ref->{url},
$ticket,
                                                          login  =>
$info_ref->{login},
                                                          passwd =>
$info_ref->{passwd});
                }
                else
                {
                    $upload_client = new LS_UploadClient($info_ref->{url},
$ticket);
                }

                if (!defined($upload_client))
                {
                    LS_LogPrint "Failed to get upload session\n";
                    exit(1);  # FIXME upload the pictures anyway.
                }
            }

            $upload_client->PerformSyncRequest(\@tagged_guids, \@sync_guids);
            $list_ref_to_fetch = \@sync_guids;
            LS_LogPrint sprintf "Tagged guids from camera\n%s\n", Dumper
\@tagged_guids;
            LS_LogPrint sprintf "Sync guids from data base\n%s\n", Dumper
\@sync_guids;
#            print Dumper \@sync_guids;
        }


        my($guid_ref, $key);
        foreach $guid_ref (@$list_ref_to_fetch)
        {

            if (!&CamspoolGetPicRec($uichan, $info_ref, $guid_ref, $ticket))
            {
                LS_LogPrint "Failed to get picture record for
$guid_ref->{id}\n";
                $done = 1;
                last;
            }
            else
            {
                $picture_count++;
            }
        }
```

19

```perl
        #
        # if we synced with the database and nothing
        # failed, then pull off ancillary files
        # of those pictures not needing data
        #
        if (exists($info_ref->{url}) && (!$done) && ($#tagged_guids >
$#sync_guids))
            {
                if (!&CamspoolResolveAncFiles($uichan, $ticket, $info_ref,
\@tagged_guids, \@sync_guids))
                {
                    $done = 1;
                }
            }

    }

    my $session_time = time - $session_start_time;
    my $report = sprintf "Transfered %d pictures %d files %d bytes in %d
seconds ",
                        $picture_count,
                        $info_ref->{session_count},
                        $info_ref->{byte_count},
                        $session_time;

    if (($info_ref->{session_count} > 0) && ($session_time > 0))
    {
        $report .= sprintf "%d bytes/sec\n",
int($info_ref->{byte_count}/$session_time);
    }
    else
    {
        $report .= "\n";
    }

    LS_LogPrint $report;
}

#
#   CamspoolGetPicRec
#
#   This function gets a picture "record" from the camera and
#   spools it to disk as specified in the $info_ref->{local_dir}.
#   A picture recond consists of  some number of "comparments"
#   of image data, and a ancillary file.
#
#   In order for the progress bar on the phone to behave properly,
#   I need to tell the camera the start and end percents the
#   compartment is of the whole picture.
#
sub CamspoolGetPicRec
{
    my($uichan, $info_ref, $guid_ref, $ticket) = @_;

    #
    # start by accumulating the total bytes for all of the compartments
    # and create a list of just comparment tags in $guid_ref.
    #
```

```perl
    my $compartment_list_ref = $guid_ref->{part};
    my $compartment_ref;

    my $total_compartment_bytes = 0;
    foreach $compartment_ref (@$compartment_list_ref)
    {
        $total_compartment_bytes += $compartment_ref->{tobyte} -
$compartment_ref->{frombyte};
    }

    LS_LogPrint sprintf "%s has %d compartments for %d bytes\n",
                        $guid_ref->{id},
                        ($#$compartment_list_ref + 1),
$total_compartment_bytes;

    my $compartments_fetched = 0;
    my $percent_complete     = 0;

    my $compartment_end_percent;
    my $compartment_percent;

    my $saved_filename;
    my $job_filename;
    my $compartment_bytes;

    my ($bytes_read, $bytes_2_read);
    foreach $compartment_ref (@$compartment_list_ref)
    {
        $compartment_bytes   = $compartment_ref->{tobyte} -
$compartment_ref->{frombyte};

        $compartment_percent =
int(($compartment_bytes/$total_compartment_bytes)*100);

        $compartment_end_percent = $percent_complete + $compartment_percent;

        LS_LogPrint sprintf "    Fetching %s with %-6d bytes %2d%% - %2d%%\n",
                        "$guid_ref->{id}.pp$compartment_ref->{id}",
                        $compartment_bytes,
                        $percent_complete,
                        $compartment_end_percent;

        $saved_filename = &CamspoolComputeUniqueFileName($info_ref,
                                                    $guid_ref->{id},
sprintf(".pp%s",$compartment_ref->{id}));

        ($bytes_read, $bytes_2_read) = $uichan->GetPic($guid_ref->{id},
                                    $saved_filename,
                                    $compartment_ref->{id},
                                    startPercent => $percent_complete,
                                    endPercent   =>
$compartment_end_percent,);

        if (($bytes_read <= 0) || ($bytes_read != $bytes_2_read))
        {
            LS_LogPrint "    failed to get part $compartment_ref->{id} for
$guid_ref->{id} $bytes_read $bytes_2_read\n";
```

```perl
                last;
        }

        LS_LogPrint "    Successfully fetched
$guid_ref->{id}.pp$compartment_ref->{id}\n";

        #
        # Write backend "job" file here.
        #
        $job_filename = &CamspoolComputeUniqueFileName($info_ref,
$guid_ref->{id}, "");

        &CamspoolWriteTagFile($job_filename,
                                guid           => $guid_ref->{id},
                                part           => $compartment_ref->{id},
                                savedFilename  => $saved_filename,
                                ticket         => $ticket,
                                type           => "image/x-lspp",);

        $compartments_fetched++;

        $info_ref->{session_count}++;
        $info_ref->{byte_count} += $compartment_bytes;

        #
        # attempt to set state for compartment successfully sent
        #
        if (!$uichan->SetPhotoState($guid_ref->{id}, "SENT",
$compartment_ref->{id}))
        {
                LS_LogPrint "    failed to set photo state to SENT
$compartment_ref->{id} for $guid_ref->{id}\n";
                last;
        }

        $percent_complete += $compartment_percent;

    }


    #
    # we will always have to grab the anc, so we don't return
    # successfully until we we've got it, wrote the local ticket and tag files
    # and finally tell the camera we are "DONE"
    #
    my $rval = 0;

    if ($compartments_fetched == ($#$compartment_list_ref + 1))
    {
        $rval = &CamspoolFetchAnc($uichan, $guid_ref->{id}, $ticket,
$info_ref);
    }

    return($rval);
}

#
#   CamspoolFetchAnc
```

22

```perl
#
#  This function will fetch an ancillary file from the camera
#  and store it locally and create the necessary job.
#  This function will also delete the picture or set the
#  state to "DONE" since the Anc file is the last thing
#  we deal with for a picture.
#
sub CamspoolFetchAnc
{
    my ($uichan, $guid, $ticket, $info_ref) = @_;

    my $saved_filename = &CamspoolComputeUniqueFileName($info_ref, $guid,
".anc");

    LS_LogPrint "    Fetching $guid.anc\n";

    my $rval = 0;

    my ($bytes_read, $bytes_2_read) = $uichan->GetPic($guid,
                                               $saved_filename,
                                               "anc");

    if (($bytes_read > 0) && ($bytes_read == $bytes_2_read))
    {
        my $job_filename = &CamspoolComputeUniqueFileName($info_ref, $guid,
"");

        &CamspoolWriteTagFile($job_filename,
                        guid           => $guid,
                        savedFilename  => $saved_filename,
                        ticket         => $ticket,
                        type           => "image/x-lsanc",);

        $info_ref->{session_count}++;
        $info_ref->{byte_count} += $bytes_read;

        if (exists($info_ref->{del_photo}))
        {
            if (!$uichan->DeletePhoto($guid))
            {
                LS_LogPrint "    failed to delete photo $guid\n";
            }
            else
            {
                LS_LogPrint "    Successfully fetched $guid.anc\n";
                $rval = 1;
            }
        }
        else
        {
            if (!$uichan->SetPhotoState($guid, "DONE", 0))
            {
                LS_LogPrint "    failed to set photo state to DONE for
$guid\n";
            }
            else
            {
                LS_LogPrint "    Successfully fetched $guid.anc\n";
```

23

```
                    $rval = 1;
                }
            }
        }
        else
        {
            LS_LogPrint "      failed to get anc for $guid\n";
        }

        return($rval);
}
#
#   This function will write a job file ending with ".tag".
#   Since the camspool backend is periodically looking for
#   files ending with ".tag" we write a ".tmp" first and
#   then rename it when it has been completely written.
#
sub CamspoolWriteTagFile
{
        my($job_filename, %args) = @_;

        if (!open(FILE, ">$job_filename.tmp"))
        {
            LS_LogPrint "Failed to write tmp file, $job_filename.tmp: $!\n";
            exit(1);
        }

        my $key;

        foreach $key (keys %args)
        {
            print FILE "$key $args{$key}\n";
        }

        if (!close(FILE))
        {
            LS_LogPrint "Failed to close tmp file, $job_filename.tmp: $!\n";
            exit(1);
        }

        if (!rename("$job_filename.tmp", "$job_filename.tag"))
        {
            LS_LogPrint "Failed to rename $job_filename.tmp to $job_filename.tag:
$!\n";
            exit(1);
        }

}


#
#   CamspoolResolveAncFiles
#
#   This function will fetch any ancillary files that might need to be loaded.
#   The idea is that users can "send" multiple emails of the picture or
#   change "Meta" data anytime.  The database sync, however, tells what
#   data is already uploaded. So given the two references to guid lists.
#   we will fetch ancillary data for any picture in the list fetched from
#   from the camera that was not part of the list fetched from the server.
```

24

```perl
#   The function will return 1 only if all ancillary files are
#   successfully fetched; 0 is returned otherwise.
#
sub CamspoolResolveAncFiles
{
    my ($uichan, $ticket, $info_ref, $full_guid_list_ref,
$synced_guid_list_ref) = @_;

    my %sync_guids;
    my $guid_ref;

    #
    # make a hash whose keys list the guids of the already synced pictures
    #
    foreach $guid_ref (@$synced_guid_list_ref)
    {
        $sync_guids{$guid_ref->{id}} = 1;
    }

    my @remaining_guids = ();

    #
    # figure out which guids in the full list are not
    # in the synced list
    #
    foreach $guid_ref (@$full_guid_list_ref)
    {
        if (!exists($sync_guids{$guid_ref->{id}}))
        {
            if (!&CamspoolFetchAnc($uichan, $guid_ref->{id}, $ticket,
$info_ref))
            {
                return(0);
            }
        }
    }

    return(1);
}


#
# CamspoolComputeUniqueFileName
#
# The trick is that we need to create a unique filename for each file written
# by a deamon into its "local_dir". Since we might be waiting to time out on
# camera's connection, while user "retries" we cannot use just the guid.
# In addition we want the filenames to "sort" alphabetically and represent
# the linear time they came in.
#
# On any given machine, time() returns a monatonically increasing number, but
# many tcp connections can happen in a single second.  Therefore the Listener
# increments a number for any connection accepted.  Also a counter is kept
# for the number of files in a session.  Lastly, there may be many dameons
dumping
# to the same directory; so we include the port number.  We add the guid
# for good measure.
#
sub CamspoolComputeUniqueFileName
```

```perl
{
    my ($info_ref, $guid, $ext) = @_;

    return sprintf "%s/%08x_%04x_%08x_%08x_%s%s",
                    $info_ref->{local_dir},
                    time,
                    $info_ref->{port},
                    $info_ref->{session_id},
                    $info_ref->{session_count},
                    $guid,
                    $ext;
}


# LS_Uichan.pm
#==============================================================================
v
#
####BSTDHDR####
#
#==============================================================================
=
#
####DESCRIPTIONBEGIN####
#  AUTHOR(S):      Matthew H. Gerlach
#  PURPOSE:        A Lightsurf Uichan client object
#  DESCRIPTION:
#      This module implements an object orieted interfact to Uichan
#      client code.
#
#
####DESCRIPTIONEND####
#==============================================================================
=
#
####COPYRIGHTBEGIN####
#
#
#
####ESTDHDR####
#==============================================================================
^



package LS_Uichan;

use IO::Select;
use XML::Simple;
use POSIX;
use MD5;
use Data::Dumper;



my $CamXMLVer = "<Ver>10</Ver>";

my $InterReadTimeout = 120;              # a two minute timeout for between reads
```

26

```perl
my $WorstCaseByteTransferRate = 100; # figure worst case transfer 100bytes/sec

#
#   This is the constructor for a uichan object.
#   It is expecting as input a IO::Handle that is usually
#   a connected TCP socket.  Communication between this uichan client
#   and a "camera" requires non blocking communication to support
#   timing out on responses.  In order to save a system call to flush
#   outgoing data on the socket, we set the socket to autoflush.  This
#   is fine because we buffer messages in application memory before
#   writing them.
#
sub new
{
    my($type, $sock) = @_;

    my $uichan = { "sock" => $sock, };

    fcntl($sock, F_SETFL(), O_NONBLOCK());

    autoflush $sock 1;

    return bless $uichan, $type;
}

#
#   AuthCamera
#
#   This method attempts to authenticate a camera.
#   This operation requires performing a "WriteRegistry"
#   uichan command to set the challenge and a "ReadRegistry"
#   operation to fetch the MD5'd output.  If the registry read
#   returns the expected data based on the challenge and the
#   presumed shared secret key, we consider the camera authenticated.
#
#
sub AuthCamera
{
    my $this = shift;
    my (%RegistryHash) = ();

    my ($challeng, $i);
    my $mysecret   = "gerry";

    #
    # Make a random challeng
    #
    for ($i = 0; $i < 8; $i++)
    {
        $challeng .= sprintf "%02x", int(rand(256));
    }

    $RegistryHash{root}       = "2";
    $RegistryHash{subkey}     = "";
    $RegistryHash{type}       = "UTF-8";
    $RegistryHash{volatile}   = "true";
    $RegistryHash{name}       = "W";
    $RegistryHash{value}      = $challeng;
```

27

```perl
    #
    # Send challeng as a Write Registry operation
    #
    if (!$this->WriteRegistry(\%RegistryHash))
    {
        main::LS_LogPrint "failed to write registry\n";
        return(0);
    }

    $RegistryHash{name}        = "U";

    #
    # To get response to challenge involves a registry read
    #
    my $challeng_resp = $this->ReadRegistry(\%RegistryHash);

    if (!defined($challeng_resp))
    {
        main::LS_LogPrint "failed to read registry\n";
        return(0);
    }

#   printf "challeng %s challeng_resp %s\n",$challeng,  $challeng_resp;

    my $md5 = new MD5;

    $md5->add($challeng, $mysecret);

    my $digest = $md5->digest();

    #
    # The challenge response number comes over the wire(less)
    # as hex encoded ascii so we we create such a string for
    # comparison.
    #
    my ($hex_digest, $byte, @bytes);
    @bytes = unpack ("C*", $digest);
    foreach $byte (@bytes)
    {
        $hex_digest .= sprintf("%02X", $byte);
    }

    if ($hex_digest eq $challeng_resp)
    {
        return(1);
    }
    else
    {
        return(0);
    }
}

#
# Fetching the ticket involves a single registry read.
#
sub GetTicket
{
    my($this) = @_;
```

28

```perl
    my (%RegistryHash) = ();
    $RegistryHash{root}        = "2";
    $RegistryHash{subkey}      = "";
    $RegistryHash{type}        = "UTF-8";
    $RegistryHash{volatile}    = "true";
    $RegistryHash{name}        = "F";

    return $this->ReadRegistry(\%RegistryHash) ;
}

sub GetGuids
{
    my($this, $guid_type, $list_ref) = @_;

    my $fdir = $this->GetFDir("/photo");

    if (!defined($fdir))
    {
        main::LS_LogPrint "GetTaggedGuids: FDir failed\n";
        return 0;
    }

    my $filetag = $fdir->{file};

#   print Dumper $filetag;

    # start by putting all filenames in @guids
    my @guids = keys %$filetag;

    #
    # now look for any files ending in .tag or .snt, since
    # a .snt isn't really "done".
    #
    if ($guid_type eq "tagged")
    {
        @guids = grep {s/\.tag$|\.snt$//} @guids;
    }
    elsif ($guid_type eq "all")
    {
        @guids = grep {s/\.tag$|\.snt$|\.loc$|\.don$//} @guids;
    }
    else
    {
        die "Invalid guid_type passed to GetGuids: $guid_type\n";
    }

#   print "GetTaggedGuids $#guids\n";

    my($guid, $file_name, $ext, $guid_ref, $compartment_array_ref);

    foreach $guid (sort @guids)
    {
#       print "    $guid\n";

        $compartment_array_ref = [];

        $guid_ref = { "id" => $guid };
```

29

```perl
        $guid_ref->{part} = $compartment_array_ref;

        foreach $ext (".pp1", ".pp2", ".pp3")
        {
            $file_name = "$guid$ext";

            if (exists($filetag->{$file_name}))
            {
                if ($filetag->{$file_name}{fsize} > 0)
                {
                    $compartment_ref = {
                                "id" => substr($ext, -1, 1),
                                "frombyte" => 0,
                                "tobyte"   => $filetag->{$file_name}{fsize}
                    };
                    push @$compartment_array_ref, $compartment_ref;
                }
                else
                {
                    main::LS_LogPrint "WARNING: zero length compartment
$file_name\n";
                }
            }
        }


        push @$list_ref, $guid_ref;
    }

    return 1;
}

sub GetFDir
{
    my($this, $dir) = @_;

    my $req = $CamXMLVer .
                "\n<CamFDir><dir>$dir</dir></CamFDir>\n";

    $this->{sock}->print($req);

    return $this->GetXmlResponse("</CamFDirR>");
}

#
#  This function handles the response from a uichan client request
#  that results in a file transfer of data.  Basically, the data
#  is surrounded by the XML response.  The last tag before the
#  the data is <size>.  The data begins immediately afer the
#  </size>.  After the data comes the </bin>, and then the actual
#  response end tag.
#
sub GetFileResponse
{
    my($this, $local_filename, $reply_end_tag) = @_;

    my $resp = $this->Expect($InterReadTimeout, $InterReadTimeout, "<\/size>",
$reply_end_tag);
```

30

```perl
        if (!defined($resp))
        {
            main::LS_LogPrint "GetFileResponse: failed to get file size info\n";
            close(FILE);
            unlink($local_filename);
            return(-1, -1);
        }

        if ($resp =~ /$reply_end_tag$/)
        {
            main::LS_LogPrint "GetFileResponse: bad response $resp\n";
            close(FILE);
            unlink($local_filename);
            return(-1, -1);
        }

        # now we try to pull out just the decimal representation of the number of
bytes
        # in the file

        $resp =~ s/.*<size>//; # strip out everything in front of number of bytes;
        $resp =~ s/<\/size>//; # strip out everything after the number of bytes;

        my $bytes_2_read = $resp;

        my ($bytes_read, $buf);

        $bytes_read = $this->ReadBytes(\$buf, $bytes_2_read,
                                        ($bytes_2_read/$WorstCaseByteTransferRate),
$InterReadTimeout);

        if ($bytes_read != $bytes_2_read)
        {
            main::LS_LogPrint sprintf "GetFileResponse: got wrong number of bytes:
%d != %d\n",
                                        $bytes_read, $bytes_2_read;
            close(FILE);
            unlink($local_filename);
            return(0, $bytes_2_read);
        }

        if (!open(FILE, ">$local_filename"))
        {
            main::LS_LogPrint "GetFileResponse: failed to open local file,
$local_filename: $!\n";
            return(0, $bytes_2_read);
        }

    binmode(FILE);

    my $bytes_written = syswrite(FILE, $buf, $bytes_read);

    if ($bytes_written < $bytes_read)
    {
        main::LS_LogPrint "GetFileResponse: failed write data all data
$bytes_written < $bytes_read: $!\n";
        close(FILE);
        unlink($local_filename);
```

31

```perl
        return(0, $bytes_2_read);
    }

    if (!close FILE)
    {
        main::LS_LogPrint "GetFileResponse: failed to close file,
$local_filename: $!\n";
        return(-1, -1);
    }

    $resp = $this->Expect($InterReadTimeout, $InterReadTimeout,
$reply_end_tag);

    if (!defined($resp))
    {
        main::LS_LogPrint "GetFileResponse: failed to get ending xml\n";
        return(0, $bytes_2_read);
    }

    return($bytes_written, $bytes_2_read);
}

sub GetFile
{
    my($this, $remote_filename, $local_filename) = @_;

    my $req =
"$CamXMLVer\n<CamGetFile>\n<name>$remote_filename</name></CamGetFile>\n";

    $this->{sock}->print($req);

    return ($this->GetFileResponse($local_filename,"</CamGetFileR>"));
}

sub TakePic
{
    my($this) = @_;

    my $req = "$CamXMLVer\n" .
            "<CamTakePicture>\n" .
            "</CamTakePicture>\n";

    $this->{sock}->print($req);

    my $xml = $this->GetXmlResponse("</CamTakePictureR>");

    if (defined($xml))
    {
        return(1);
    }
    else
    {
        return(0);
    }
}

sub HangupServer
{
```

32

```perl
    my($this) = @_;

    my $req = "$CamXMLVer\n" .
            "<CamHangupServer>\n" .
            "</CamHangupServer>\n";

    $this->{sock}->print($req);

    my $xml = $this->GetXmlResponse("</CamHangupServerR>");

    if (defined($xml))
    {
        return(1);
    }
    else
    {
        return(0);
    }
}
sub CallServer
{
    my($this) = @_;

    my $req = "$CamXMLVer\n" .
            "<CamCallServer>\n" .
            "</CamCallServer>\n";

    $this->{sock}->print($req);

    my $xml = $this->GetXmlResponse("</CamCallServerR>");

    if (defined($xml))
    {
        return(1);
    }
    else
    {
        return(0);
    }
}

sub SetSoundState
{
    my($this, $state) = @_;

    my $req = "$CamXMLVer<CamSetSoundState>$state</CamSetSoundState>";

    $this->{sock}->print($req);

    my $xml = $this->GetXmlResponse("</CamSetSoundStateR>");

    if (defined($xml))
    {
        return(1);
    }
    else
    {
        return(0);
```

33

```perl
    }

}

sub GetPic
{
    my($this, $guid, $local_filename, $type, %args) = @_;

#    main::LS_LogPrint "GetPic($guid, $local_filename)\n";

    my $key;

    my $req = "$CamXMLVer\n" .
            "<CamGetPicture>\n" .
                "<name>$guid</name>" .
                "<type>$type</type>";

    if (defined($args{width}))
    {
        $req .=     "<width>$args{width}</width>";
    }
    else
    {
        $req .=     "<width></width>";
    }

    if (defined($args{height}))
    {
        $req .=     "<height>$args{height}</height>";
    }
    else
    {
        $req .=     "<height></height>";
    }

    if (defined($args{depth}))
    {
        $req .=     "<depth>$args{depth}</depth>";
    }
    else
    {
        $req .=     "<depth></depth>";
    }
    if (defined($args{color}))
    {
        $req .=     "<color>$args{color}</color>";
    }
    else
    {
        $req .=     "<color></color>";
    }

    if (defined($args{startPercent}))
    {
        $req .=     "<startPercent>$args{startPercent}</startPercent>";
    }
```

34

```perl
    if (defined($args{endPercent}))
    {
       $req .=    "<endPercent>$args{endPercent}</endPercent>";
    }

        $req .= "</CamGetPicture>\n";

    $this->{sock}->print($req);

    return ($this->GetFileResponse($local_filename,"<\/CamGetPictureR>"));

}

sub WriteRegistry
{
    my($this, @reg_hashes) = @_;

    my $req = "$CamXMLVer\n<CamWriteRegistryValue>\n";

    my $reg_hash;

    foreach $reg_hash (@reg_hashes)
    {
        $req .= "<registry>\n" .
                "<name>$reg_hash->{name}</name>\n" .
                "<registryType>$reg_hash->{type}</registryType>\n" .
                "<root>$reg_hash->{root}</root>\n" .
                "<subkey>$reg_hash->{subkey}</subkey>\n" .
                "<value>$reg_hash->{value}</value>\n" .
                "<volatile>$reg_hash->{volatile}</volatile>\n" .
                "</registry>\n";
    }

    $req .= "</CamWriteRegistryValue>\n";

    $this->{sock}->print($req);

    my $xml = $this->GetXmlResponse("</CamWriteRegistryValueR>");

    if (defined($xml))
    {
        return(1);
    }
    else
    {
        return(0);
    }

}

sub ReadRegistry
{
    my($this, $reg_hash) = @_;

    my $req = "$CamXMLVer\n<CamReadRegistryValue>\n";

    $req .= "<registry>\n" .
                "<name>$reg_hash->{name}</name>\n" .
```

35

```perl
                "<registryType>$reg_hash->{type}</registryType>\n" .
                "<root>$reg_hash->{root}</root>\n" .
                "<subkey>$reg_hash->{subkey}</subkey>\n" .
                "<volatile>$reg_hash->{volatile}</volatile>\n" .
            "</registry>\n";

    $req .= "</CamReadRegistryValue>\n";

    $this->{sock}->print($req);

    my $xml = $this->GetXmlResponse("</CamReadRegistryValueR>");

    if (defined($xml))
    {
        #print Dumper($xml->{registry});

        return($xml->{registry}{value});
    }
    else
    {
        return(undef);
    }
}

sub SetPhotoState
{
    my( $this, $guid, $state, $stateDetailed ) = @_;

    my $req = "$CamXMLVer\n" .
            "<CamSetPhotoState>\n" .
                "<name>$guid</name>\n" .
                "<stateType>1</stateType>\n" .
                "<photoState>$state</photoState>\n" .
                "<photoStateDetail>$stateDetailed</photoStateDetail>\n" .
            "</CamSetPhotoState>\n";

    $this->{sock}->print($req);

    my $xml = $this->GetXmlResponse("</CamSetPhotoStateR>");
    if (defined($xml))
    {
        #print Dumper($xml->{registry});

        return(1);
    }
    else
    {
        return(0);
    }

}

sub DeletePhoto
{
    my ($this, $guid) = @_;

    my $req = "$CamXMLVer<CamPDel><name>$guid</name></CamPDel>\n";
```

36

```perl
        $this->{sock}->print($req);

        my $xml = $this->GetXmlResponse("</CamPDelR>");
        if (defined($xml))
        {
            #print Dumper($xml->{registry});

            return(1);
        }
        else
        {
            return(0);
        }
}
#
# ReadBytes
#
# This is a bit of a hairy method that perfoms the actual reading of
# bytes off of the wire.  This function gets passed two time outs.
# One time out for the over all number of bytes to be read, and another
# "inter read" timeout.  By having two timeouts this function can be used
# to effeciently read large buffers, but timeout appripriately if line
# appears dead because no bytes are trickling in.
#
# This function makes the assumption that the IO::Handle has be set
# to non-blocking by the constructor.  In addition this function uses
# the "read" method on the IO::Handle instead of the "sysread" method.
# By using "read", we are taking advantage of perl's buffered io streams.
# Doing so dramatically limits the number of times this application traps
# to the Unix Kernel.  As it turns out due to Uichan's message usage
# of the IO stream, we essentially have only two system calls for
# each message, the select checking for available bytes, and the buffered
# "read" which grabs whatever bytes are availble from the kernel,
# but delivers what we ask for.
#
sub ReadBytes
{
    my($this, $buf_ref, $bytes_to_read, $total_timeout, $inter_read_timeout) =
@_;
    my($sock, $rval, $buf, $time_left, $timeout, @ready);

    $sock = $this->{sock};
    my $bytes_read = 0;

    my $start_time = time;

    while ($bytes_read < $bytes_to_read)
    {
        $rval = $sock->read($$buf_ref,
                            ($bytes_to_read - $bytes_read), $bytes_read);

        if (!defined($rval))
        {
            if ($! == EAGAIN())
            {
                $time_left = $total_timeout - (time - $start_time);

                if ($time_left <= 0)
```

```perl
            {
                main::LS_LogPrint "ReadBytes: Total timeout\n";
                last;
            }

            my $selObj = IO::Select->new();
            $selObj->add($sock);

            if ($time_left < $inter_read_timeout)
            {
                @ready =  $selObj->can_read($time_left);
            }
            else
            {
                @ready =  $selObj->can_read($inter_read_timeout);
            }
            if ($#ready < 0)
            {
                main::LS_LogPrint "ReadBytes: select timed out\n";
                last;
            }
        }
        else
        {
            main::LS_LogPrint sprintf "ReadBytes bad socket read: %d:
$!\n", $!;
            last;
        }
    }
    elsif ($rval > 0)
    {
        $bytes_read += $rval;
    }
    elsif ($rval == 0)
    {
        main::LS_LogPrint "socket closed\n";
        last;
    }
    else
    {
        main::LS_LogPrint "weird socket rval $rval\n";
        last;
    }
}

    return($bytes_read);
}

sub Empty
{
    my($this) = @_;

    my $rval;
    my $buf;

    my $sock = $this->{sock};

    while (1)
```

38

```perl
    {
        $rval = $sock->read($buf, 1024);

        last if ((!defined($rval)) || ($rval <= 0));
    }
}


#
# Expect
#
# This method is based on the Tcl extension, "Expect" by
# Don Libes.  The idea is that this function reads the stream
# looking for the "Expected" patterns to match the end of the
# stream.  If one of the "Expected" matches occurs, the entire
# buffer is return.  undef is returned on timeout.
#
sub Expect
{
    my($this, $total_timeout, $inter_read_time, @matches) = @_;

    my($buf, $byte, $time_left, $match);

    my $start_time = time;

    while (1)
    {
        $time_left = $total_timeout - (time - $start_time);

        if ($time_left <= 0)
        {
            main::LS_LogPrint "Expect: ran out of time\n";
            return(undef);
        }

        if ($this->ReadBytes(\$byte, 1, $time_left, $inter_read_time) != 1)
        {
            main::LS_LogPrint "Expect: read timed out\n";
            return(undef);
        }

        $buf .= $byte;

        foreach $match (@matches)
        {
            if ($buf =~ /$match$/)
            {
                return($buf);
            }
        }
    }
}


#
# GetXmlResponse
#
# This routine tries to fetch an Uichan Xml Response
# from the other end.  If successful, this function will
```

39

```perl
# return a hash produced by XML::Simple of the XmlDocument.
#
sub GetXmlResponse
{
    my ($this, $docEndTag) = @_;
    my $resp = "";;

    $resp = $this->Expect($InterReadTimeout, $InterReadTimeout, $docEndTag);

#   print "GetXmlReponse: response\n$resp\n--\n";

    if (!defined($resp))
    {
        main::LS_LogPrint "GetXmlResponse: Expect failed\n";
        return(undef);
    }

    if (!($resp =~ s/$CamXMLVer//))
    {
        main::LS_LogPrint "GetXmlResponse: failed to see xml version
header:\n$resp\n";
        return(undef);
    }


    my $xml = XMLin($resp);

    if (!defined($xml->{status}))
    {
        main::LS_LogPrint "GetXmlResonse: no status in resp\n";
        return(undef);
    }
    elsif ($xml->{status} ne "0")
    {
        main::LS_LogPrint "GetXmlResonse: bad status in response:
$xml->{status}\n";
        return(undef);
    }
    return $xml;

}
1;


__END__


=head1 NAME

Uichan - an object that implements Uichan Client functionality

=head1 SYNOPSIS

    use LS_Uichan;

    my $uichan = new Uichan($io_handle); # $io_handle is an open/connected
IO::Handle,
                                         # usually IO::Socket::INET
```

40

```perl
    if (!$uichan->AuthCamera())
    {
        die "Could not Auth Camera\n";
    }

    print "Successfully Authenticated Camera\n";

    my $ticket = $uichan->GetTicket();

    if (!defined($ticket))
    {
        die "Could not get Ticket\n";

    }

    print "ticket = $ticket\n";

    my $fdir = $uichan->GetFDir("/photo");

    if (!defined($fdir))
    {
        die "Could not read dir /photo\n";
    }

    if
(!$uichan->GetFile("/photo/ls_00200020_00000016_00780005apd_00000005.ppf",
                        "./ls_00200020_00000016_00780005apd_00000005.ppf"))
    {
        die "failed to get file\n";
    }

    print "successfully got file\n";
```

=head1 DESCRIPTION

This module implements an object oriented interface to client functionality
of a Lightsurf Uichan client.

=head1 CONSTRUCTOR

=over 4

=item new (io_handle)

Creates an C<LS_Uichan> object. The constructor takes one option,
a referernce to an opened IO::Handle. The constructor will the handle to
non-blocking mode to allow timeing out on responses.

At the moment only an IO::Socket::INET has actually been used with this
object.

=back

=head2 METHODS

=item AuthCamera()

Attempts to authenticate the camera.   Returns 1 on success, 0 otherwise.

=item GetTicket()

Returns a scalar representing the ticket number, undef on failure.

=item GetGuids(guid_type, list_ref)

This function attempts asks the uichan server for a list of guids.
The guid_type should be "tagged" or "all" for the guids that have been
tagged for tranmission or all guids, respectively. If the request is
successful 1 is returned, 0 otherwise. The actual data from
the response gets shifted into the passed reference to a list. The elements
shifted in are hash references which have two keys: id and part. The value
of id is the guid of the photo, and the key, part, is a reference to a list
of hashes describing compartments. Each comparment hash has three keys:
id, frombyte and tobyte. The id is the compartment id (e.g. 1, 2, 3),
and the frombyte will always be 0, and the tobyte is the length of the
compartment. The decision for the wierd structure is that it matches
the structure of LS_UploadClient::PerformSyncRequest().

=item GetFDir(remote_dir)

This function requests a listing of the passed in directory. If the
request fails, undef is returned. If successful, the parsed xml response
is returned.

=item GetFile(remote_filename, local_filename)

This function attempts to fetch the remote_filename and write to the
local_filename. Returns array ($bytes_written, $total_in_file). When
completely successful $bytes_written will be equal to $total_in_file, and both
will be greater than failure. When a catastrophic failure occurs,
$bytes_written will be -1. If a subset of the file was fetched, $bytes_written
will be greater than 0 and less than $total_in_file.

=item GetPic($guid, $local_filename, $type, %args)

This function attempts to fetch the picture, guid, and write it to
local_filename. $type should be either the comparment number (e.g. 1, 2, 3
...),
"full", "anc", "alien_preview", "generic", or "png_preview".

%args is a hash of openional named arguments. The supported named arguments
are width, height, depth, color, startPercent, and endPercent.

This method returns an array as described for the GetFile() method.

=item SetPhotoState($guid, $state, $stateDetailed)

This method attempts to set the state and detailed stated of the given
guid. 1 is returned on success, 0 othersize.

=item DeletePhotot($guid)

This method is delete the requested guid. 1 is returned on success, 0
otherwise.

42

=item TakePic()

This function will request the camera to take a picture. 1 is returned
if the request was successful, 0 otherwise. The picture taken will not
actually show up in the filesystem until some time after the response.

=item SetSoundState(state)

This function will set of sound generation. State is "1" to enable sounds,
and state is 0 to disable sounds.

=item CallServer

This function will request the camera to make a connection to a server.
It returns 1 if the command was accepted, 0 otherwise. A successful
return does not imply a successful connection to the server, just
that the camera will try. When the camera successfully connects,
an appropriate event will be sent on the event channnel. Subsequent
CallServer commands should not be sent unless a "Server Done"
event has been received.

=item HangupServer

This function will ask the camera to hangup its connection to
a server. 1 is returned if the command was accepted, 0 otherwise.
The actual connection should not be considered down until a
"Server Done" event arrives on the event channel

=item Empty()

This function "empties" any data in the receive buffer of the socket and
throws the
data away. It is usually a good idea to call this function to promote a
"clean"
closing of the socket.

=back

=head1 SEE ALSO

L<Socket>, L<IO::Socket>

=head1 AUTHOR

Matthew H. Gerlach
mgerlach@lightsurf.com

=head1 COPYRIGHT

####COPYRIGHTBEGIN####

    (c) Copyright 1999, 2000 Lightsurf Technologies, Inc.   ALL RIGHTS RESERVED.

=cut

```
#LS_UploadClient.pm
#==============================================================================
v
#
####BSTDHDR####
#
#==============================================================================
=
#
####DESCRIPTIONBEGIN####
#   AUTHOR(S):
#   PURPOSE:
#   DESCRIPTION:
#
#
####DESCRIPTIONEND####
#==============================================================================
=
#
####COPYRIGHTBEGIN####
#
#   (c) Copyright 1999, 2000 Lightsurf Technologies, Inc.   ALL RIGHTS RESERVED.
#
#
#
####COPYRIGHTEND####
#==============================================================================
=
#
####ESTDHDR####
#==============================================================================
^

package LS_UploadClient;

use strict;
#use LWP::Debug qw(+);
use LWP::UserAgent;
use XML::Simple;
use Data::Dumper;
use LS_UnixDaemonUtils;


sub new
{
    my ($type, $url_start, $ticket, %args) = @_;

    my $upload_client =
    {
        url_start => $url_start,
        ticket    => $ticket,
    };

    if (defined($args{login}))
    {
        $upload_client->{login} = $args{login};
    }
```

44

```perl
if (defined($args{passwd}))
{
    $upload_client->{passwd} = $args{passwd};
}

if (defined($args{imsi}))
{
    $upload_client->{imsi} = $args{imsi};
}
else
{
    $upload_client->{imsi} = "123";
}

if (defined($args{imei}))
{
    $upload_client->{imei} = $args{imei};
}
else
{
    $upload_client->{imei} = "123";
}

if (defined($args{pstn}))
{
    $upload_client->{pstn} = $args{pstn};
}
else
{
    $upload_client->{pstn} = "123";
}

my($get_url) =
    $upload_client->{url_start} .
    "/authenticate?handler=device&devicelogin=$upload_client->{ticket}" .
    "&camera_id=123&imsi=$upload_client->{imsi}" .
    "&imei=$upload_client->{imei}&pstn=$upload_client->{pstn}" .
    "&resource=/asst/resource_index.jsp";

my $get_agent = new LWP::UserAgent;

my $get_req = new HTTP::Request('GET', $get_url);

if (defined($upload_client->{login}) && defined($upload_client->{passwd}))
{
    $get_req->authorization_basic($upload_client->{login},
$upload_client->{passwd});
}

my $res = $get_agent->request($get_req);

if (!$res->is_success)
{
    LS_LogPrint "failed to get session id\n";
    LS_LogPrint Dumper($res);
    return(undef);
}
```

45

```perl
    my $hdrs = $res->headers;

    my ($session_id) = ($hdrs->as_string(" ") =~ /JSESSIONID=(.*?);/);
    my ($machine_id) = ($hdrs->as_string(" ") =~ /machineid=(.*?);/);

    #
    # FIXME
    # I should check for session_id and machine_id
    #
    $upload_client->{session_id} = $session_id;
    $upload_client->{machine_id} = $machine_id;

    return bless $upload_client, $type;
}

sub PerformSyncRequest
{
    my ($this, $in_list_ref, $out_list_ref) = @_;

    my $post_url = "$this->{url_start}/asst/sync_asst.jsp";

    my $post_agent = new LWP::UserAgent;
    my $post_req   = new HTTP::Request('POST', $post_url);

    if (defined($this->{login}) && defined($this->{passwd}))
    {
        $post_req->authorization_basic($this->{login}, $this->{passwd});
    }

    $post_req->header("Cookie" => "JSESSIONID=$this->{session_id}");
    $post_req->push_header(Cookie => "machineid=$this->{machine_id}");

    my ($part_info, $i);

    my $xml_req = "<?xml version=\"1.0\" ?>\n<photos>\n";

      my ($guid_ref, $compartment_array_ref, $compartment_ref);
    foreach $guid_ref (@$in_list_ref)
    {
        $part_info = "";
        $compartment_array_ref = $guid_ref->{part};
        foreach $compartment_ref (@$compartment_array_ref)
        {
            $part_info .= sprintf "     <part
id=\"%d\"><offset>0</offset><length>%d</length></part>\n",
                                        $compartment_ref->{id},
$compartment_ref->{tobyte};
        }

        if (length($part_info) > 0)
        {
            $xml_req .= "<photo id=\"$guid_ref->{id}\">\n" . $part_info .
"</photo>\n";
        }
    }

    $xml_req .= "</photos>\n";
```

46

```perl
    $post_req->content($xml_req);

    my $res = $post_agent->request($post_req);

    if (!$res->is_success)
    {
        LS_LogPrint "PerformSyncRequest post failed\n";
        LS_LogPrint Dumper($res);
        return(0);
    }

    LS_LogPrint "PerformSyncRequest post succeeded\n";
#    LS_LogPrint Dumper($res);
#    LS_LogPrint sprintf "Content = %s\n", $res->content;
    my $xml_resp = XMLin($res->content, keyattr => 'sendphoto',
                                        forcearray => ['sendphoto', 'part']);

    my $photo_list_ref = $xml_resp->{sendphoto};

    if (defined($photo_list_ref))
    {
        @$out_list_ref = @$photo_list_ref;
    }
    else
    {
        @$out_list_ref = ();
    }

#    LS_LogPrint Dumper($xml_resp);
    return(1);
}
sub UploadImageCompartment
{
    my ($this, $guid, $type, $picture_file_name, $part, $offset, $length) =
@_;

    if (!open(PICFILE, $picture_file_name))
    {
        LS_LogPrint "failed to open picture file, $picture_file_name: $!\n";
        return undef;
    }
    binmode(PICFILE);
    my @file_stat = stat(PICFILE);
    my $len        = $file_stat[7];
    my $image_data;
    sysread(PICFILE, $image_data, $len);
    close(PICFILE);

    my $post_url = "$this->{url_start}/asst/upload_asst.jsp";

    my $post_agent = new LWP::UserAgent;
    my $post_req   = new HTTP::Request('POST', $post_url);

    if (defined($this->{login}) && defined($this->{passwd}))
    {
        $post_req->authorization_basic($this->{login}, $this->{passwd});
    }
```

47

```perl
my $uniq_id = "529021". time;

my $boundary = "---------------------------$uniq_id";

$post_req->header("Cookie" => "JSESSIONID=$this->{session_id}");
$post_req->push_header(Cookie => "machineid=$this->{machine_id}");

$post_req->content_type("multipart/form-data; " .
                "boundary=$boundary\r\n");


$boundary = "--$boundary";
## Build the data sent before the image...
my($before, $end);

$before  = "$boundary\r\n";
$before .= "Content-Disposition: form-data; name=\"Image1\"; filename=\"";
$before .= "untitled" . "\"\r\n";
$before .= "Content-Type: $type\r\n";
$before .= "\r\n";

$end =   "$boundary\r\n";
$end .= "Content-Disposition: form-data; name=\"Image1guid\"";
$end .= "\r\n";
$end .= "\r\n";
$end .= "$guid";
$end .= "\r\n$boundary\r\n";

$end .= "Content-Disposition: form-data; name=\"Image1partid\"";
$end .= "\r\n";
$end .= "\r\n";
$end .= "$part";
$end .= "\r\n$boundary\r\n";

$end .= "Content-Disposition: form-data; name=\"Image1offset\"";
$end .= "\r\n";
$end .= "\r\n";
$end .= "0";
$end .= "\r\n$boundary\r\n";

$end .= "Content-Disposition: form-data; name=\"Image1length\"";
$end .= "\r\n";
$end .= "\r\n";
$end .= "$len";
$end .= "\r\n$boundary--\r\n";
# last boundary needs ending --

   my $content = $before.$image_data."\r\n".$end;

$post_req->content( $content);

LS_LogPrint "posting $guid part $part\n";
my $res = $post_agent->request($post_req);

if (!$res->is_success)
{
    LS_LogPrint "HTTP upload post failed for $guid\n";
    LS_LogPrint sprintf "%s\n", $res->content;
```

48

```perl
            return(0);
        }

        my $xml_ref = XMLin($res->content);
#       LS_LogPrint sprintf "xml response\n%s\n", Dumper $xml_ref;

        if (defined($xml_ref->{photo}) && defined($xml_ref->{photo}->{id}))
        {
            if (defined($xml_ref->{partalreadyreceived}))
            {
                LS_LogPrint "post succeeded for $guid part $part id
$xml_ref->{photo}->{id} already in db\n";
            }
            else
            {
                LS_LogPrint "post succeeded for $guid part $part id
$xml_ref->{photo}->{id}\n";
            }

            return($xml_ref->{photo}->{id});
        }
        elsif (defined($xml_ref->{error}))
        {
            LS_LogPrint "unrecoverable error from server: $xml_ref->{error}\n";
            return(0);
        }

        # If we fall through here, there was some error in the response.
        LS_LogPrint "failed response to upload post for $guid part $part\n";
        LS_LogPrint sprintf "%s\n", $res->content;
        return(0);
}

sub SetMetaData
{
        my ($this, $idtype, $id, %args) = @_;

        my($get_url) = "$this->{url_start}/asst/update_photo_asst.jsp?";

        if ($idtype eq "guid")
        {
            $get_url .= "guid=$id&";
        }
        elsif ($idtype eq "elementID")
        {
            $get_url .= "elementID=$id&";
        }
        else
        {
            LS_LogPrint "SetComments got a bad 'type' parameter: $args{type}\n";
            exit(1);
        }

        if (defined($args{title}))
        {
            $get_url .= "&name=$args{title}";
        }
```

49

```perl
        if (defined($args{comments}))
        {
            $get_url .= "&description=$args{comments}";
        }

        if (defined($args{location}))
        {
            $get_url .= "&location=$args{location}";
        }


#       LS_LogPrint "SetComments url $get_url\n";
        my $get_agent = new LWP::UserAgent;
        my $get_req   = new HTTP::Request('GET', $get_url);

        if (defined($this->{login}) && defined($this->{passwd}))
        {
            $get_req->authorization_basic($this->{login}, $this->{passwd});
        }

        $get_req->header("Cookie" => "JSESSIONID=$this->{session_id}");
        $get_req->push_header(Cookie => "machineid=$this->{machine_id}");

        my $res = $get_agent->request($get_req);

        if (!$res->is_success)
        {
            LS_LogPrint "HTTP request failed to set comments for $idtype $id\n";
            LS_LogPrint Dumper($res);
            return(0);
        }
        elsif ( $res->content !~ /<success\/>/)
        {
            LS_LogPrint "XML response failed to set comments for $idtype $id\n";
            LS_LogPrint Dumper($res);
            return(0);
        }
        else
        {
            LS_LogPrint "Sucessfully set comments for $idtype $id\n";
            return(1);
        }
}

sub ShipEmailAddrs
{
    my ($this, $idtype, $id, $email_list_ref) = @_;

    my($get_url) = "$this->{url_start}/asst/send_greeting.jsp?";

    if ($idtype eq "guid")
    {
        $get_url .= "guid=$id&";
    }
    elsif ($idtype eq "elementID")
    {
        $get_url .= "elementID=$id&";
    }
```

50

```perl
        else
        {
            LS_LogPrint "ShipEmailAddrs bad idtype $idtype\n";
            exit(1);
        }

        $get_url .= "toAddress=$email_list_ref->[0]";

        my($i);

        for ($i = 1; $i <= $#$email_list_ref; $i++)
        {
            $get_url = $get_url . ",$email_list_ref->[$i]"
        }

#       LS_LogPrint "Email list url:\n     $get_url\n";

        my $get_agent = new LWP::UserAgent;
        my $get_req   = new HTTP::Request('GET', $get_url);

        if (defined($this->{login}) && defined($this->{passwd}))
        {
            $get_req->authorization_basic($this->{login}, $this->{passwd});
        }

        $get_req->header("Cookie" => "JSESSIONID=$this->{session_id}");
        $get_req->push_header(Cookie => "machineid=$this->{machine_id}");

        my $res = $get_agent->request($get_req);

        if (!$res->is_success)
        {
            LS_LogPrint "HTTP request failed for email post of $idtype $id\n";
            LS_LogPrint Dumper($res);
            return(0);
        }
        elsif ($res->content =~ /<success\/>/)
        {
            LS_LogPrint "Sucessfully sent shipped email addrs for $idtype $id\n";
            return 1;
        }
        else
        {
            LS_LogPrint "Unknown XML response\n" . $res->content . "\n";
            return 0;
        }
}
sub GetEmailUrl
{
    my ($this, $idtype, $id, $email_list_ref) = @_;

    my($get_url) = "$this->{url_start}/asst/get_greeting.jsp?";

    if ($idtype eq "guid")
    {
        $get_url .= "guid=$id&";
    }
    elsif ($idtype eq "elementID")
```

51

```perl
    {
        $get_url .= "elementID=$id&";
    }
    else
    {
        LS_LogPrint "GetEmailUrl bad idtype $idtype\n";
        exit(1);
    }

    $get_url .= "toAddress=$email_list_ref->[0]";

    my($i);

    for ($i = 1; $i <= $#$email_list_ref; $i++)
    {
        $get_url = $get_url . ",$email_list_ref->[$i]"
    }

#    LS_LogPrint "Email list url:\n     $get_url\n";

    my $get_agent = new LWP::UserAgent;
    my $get_req   = new HTTP::Request('GET', $get_url);

    if (defined($this->{login}) && defined($this->{passwd}))
    {
        $get_req->authorization_basic($this->{login}, $this->{passwd});
    }

    $get_req->header("Cookie" => "JSESSIONID=$this->{session_id}");
    $get_req->push_header(Cookie => "machineid=$this->{machine_id}");

    my $res = $get_agent->request($get_req);

    if (!$res->is_success)
    {
        LS_LogPrint "HTTP request failed for /asst/get_greeting.jsp post of
$idtype $id\n";
        LS_LogPrint Dumper($res);
        return(undef);
    }

    my $xml_ref = XMLin($res->content);

    if (!exists($xml_ref->{shareurl}))
    {
        LS_LogPrint "GetEmailUrl got bad XML response\n$res->content\n";
        return(undef);
    }

    return($xml_ref->{shareurl});
}
1;

__END__

=head1 NAME
```

52

LS_UploadClient - an object for uploading pictures to the LightSurf Server.

=head1 SYNOPSIS

    use LS_UploadClient;

    $upload_client = new LS_UploadClient("http://www.photosurf.com",
$device_login);

    $upload_client->UploadImageCompartment($guid, $filename, $part, $offset,
$length);

    $upload_client->SetComments($guid, "My Title", "My Location", "Some
Comments");

    $upload_client->ShipEmailAddrs($guid, "foo@bar.com", "bar@foo.com");

=head1 DESCRIPTION

LS_UploadClient provides an object oriented interface to uploading/syncing
picutures with a LightSurf Server. An instance of the object can be used
to make uploads to a particular user's account. When a new account is
uploaded to, a new object must be created.

=head1 CONSTRUCTOR

=item new (url, device_login, %args)

The constructor creates an instance for an upload session to a particular
user's account. The parameters are the base url to LightSurf server and
a device login that is fetched from a camera. This device login associates
to a particular user's account. Once constructed, the object can be used
to upload many pictures into the account and/or set properities of pictures.
The constructor actually communicates to the server to fetch a session id;
so it can fail. On failure undef is returned. The constructor has optional
parameters, %args, that are passed as name => value pairs. For server
authentication, login and passwd name/values can be passed. In addition
imsi, imei, and pstn named parameters are supported.

=head1 METHODS

=item UploadImageCompartment(guid, type, filename, compartment_num, offset,
length)

This method uploads a compartment to an account on the server. It needs
the picture's globally unique id (guid), the type of comparment
(i.e. "image/jpeg" or "image/x-lspp"), the filename of the compartment,
the compartment number (e.g. "1", "2", ...). In addition the offset into
the file and length of the bytes is given. Usually the offset is 0 and the
length is the size of the file, but it could be smaller. On success the
resulting positive photo_id from the database is returned. undef is
returned if the upload failed, and should be retried. 0 is returned if
the failure case should not be retried.

=item SetMetaData(idtype, id, [optional named arguments])

53

This method will set any meta data associated with a picture. The idtype
should be "guid" or "elementID" if a picture guid or its upload elementID
is the id. The opional named arguments are title, comments, and location.
1 is returned on success, 0 on failure.

=item ShipEmailAddrs(idtype, id, list of email addresses)

This method will ask the LightSurf server to share the identified picture with
the passed reference to a list of email addresses. Like SetMetaData, the
idtype
should be "guid" or "elementID". 1 will be returned on success, 0 on failure.

=item GetEmailUrl(idtype, id, list of email addresses)

This method will ask the LightSurf to setup a picture to be shared. It
has the same parameters as ShipEmailAddrs, but its return value is different.
On success the method will return a URL that represents the shared picture.
undef is return on failure.

=cut